CSN08x14 Optional extension for lab 4: Sorting algorithms in Python

1. Understanding Sorting Algorithms

There are many well-known sorting algorithms. Some of the most famous are **Bubble sort, Merge sort, Insertion Sort, Quick sort**.

a. Watch the "battle of the algorithms" clip available at http://www.bbc.co.uk/programmes/p032rf83 (4 minutes, *needs sound* - *you may prefer to do this bit at home*), which explains bubble sort and merge sort.

   Answer the following questions:

   - How does bubble sort work?
   - How does merge sort work?
   - Which of the concepts discussed in the lecture does merge sort use?
   - Which is more efficient, bubble sort or merge sort?

b. Watch the visualisation of bubble sort and quick sort at https://youtu.be/aXXWXz5rF64 (3 minutes, no need for sound), and read through http://www.pythonschool.net/data-structures-algorithms/insertion-sort/ and http://www.pythonschool.net/data-structures-algorithms/quicksort/.

   Answer the following questions:

   - Explain how quick sort works?
   - Compare the purpose of the marker used in insertion sort with the purpose of the pivot in quick sort?
   - Which of the concepts discussed in the lecture does quick sort use?
   - Which is more efficient, quick sort or bubble sort? Why?

c. Play with the sorting algorithm animations at http://www.sorting-algorithms.com/. Watch the 4 algorithms we considered before to compare their performance for four different data examples. According to the website, one of the aims of the animation is to show that there is no single best sorting algorithm.

   Can you describe an example situation where quick sort is NOT the best sorting algorithm?

2. Quicksort in Python

Implementing quick sort in Python is not all that difficult, and many others have written such implementations already.

So, have a look at the implementations at

- http://hetland.org/coding/python/quicksort.html, http://interactivepython.org/runestone/static/pythonds/SortSearch/TheQuickSort.html and http://www.pythonschool.net/data-structures-algorithms/quicksort/.

Hackerrank also has this as a guided challenge, split into two parts:

- Quicksort1 (partition an array): https://www.hackerrank.com/challenges/quicksort1

- Quicksort2 (sort using quicksort and print each step): https://www.hackerrank.com/challenges/quicksort2

a. Based on these, implement a quick sort in Python yourself. Make sure that you understand your program fully and comment your code appropriately.
b. Test that it works – use several different lists for this purpose.
c. Write a wrapper program that takes a number n as its input, generates a random list of length n and uses quick sort to sort it.
   Hint: To do this, you will need to generate large amounts of random data. This can be done with python's random functions. To find out how to do this more efficiently than using a simple for loop, refer to https://stackoverflow.com/questions/7988494/efficient-way-to-generate-and-use-millions-of-random-numbers-in-python and
d. Adapt the program to count the number of comparisons made during the sorting process.

3. Built-in sorting in Python

Like all programming languages, Python has an efficient built-in sorting function. Python uses the timsort algorithm, which is a hybrid algorithm derived from merge sort and insertion sort. Find out more by reading the pages http://pythoncentral.io/how-to-sort-a-list-tuple-or-object-with-sorted-in-python/ and https://en.wikipedia.org/wiki/Timsort.

a) What is the difference between sort() and sorted()?
b) How do you sort in reverse (highest to lowest)?
c) Adapt your program from Exercise 4c/d to use the built-in sorting function instead of your own implementation of quick sort.